# Design Document

## High Level Approach

For the first backup of any dataset, we would copy the whole dataset all together, serving as backup-0. Then, from consequent backups (backup-1, backup-2…) would store only modifications to the dataset.
Backup can be updated either on a time-basis (a continuously running script checks every x hours whether the dataset has been modified) or a change-basis (when there is a change in the dataset, the frontend of backup modification can call our backup function).We'll assume time-basis for now since that is harder to implement, and is a superset of change-basis form. To identify if there has been a change after x hours have passed, last modified times of the files in the dataset would be compared against the last backup's timestamp. If no change is found, the dataset won't be backed-up again.
When writing a backup where only modifications are stored, we would write the directory tree of the dataset at that point in time, the list of deleted file names, the list of added file names, and the list of modified file names as metadata. The modified files and added files would be stored in respective folders. The deleted files don't need to be stored as they are not present in the new version, and would be available in the previous version's backup.

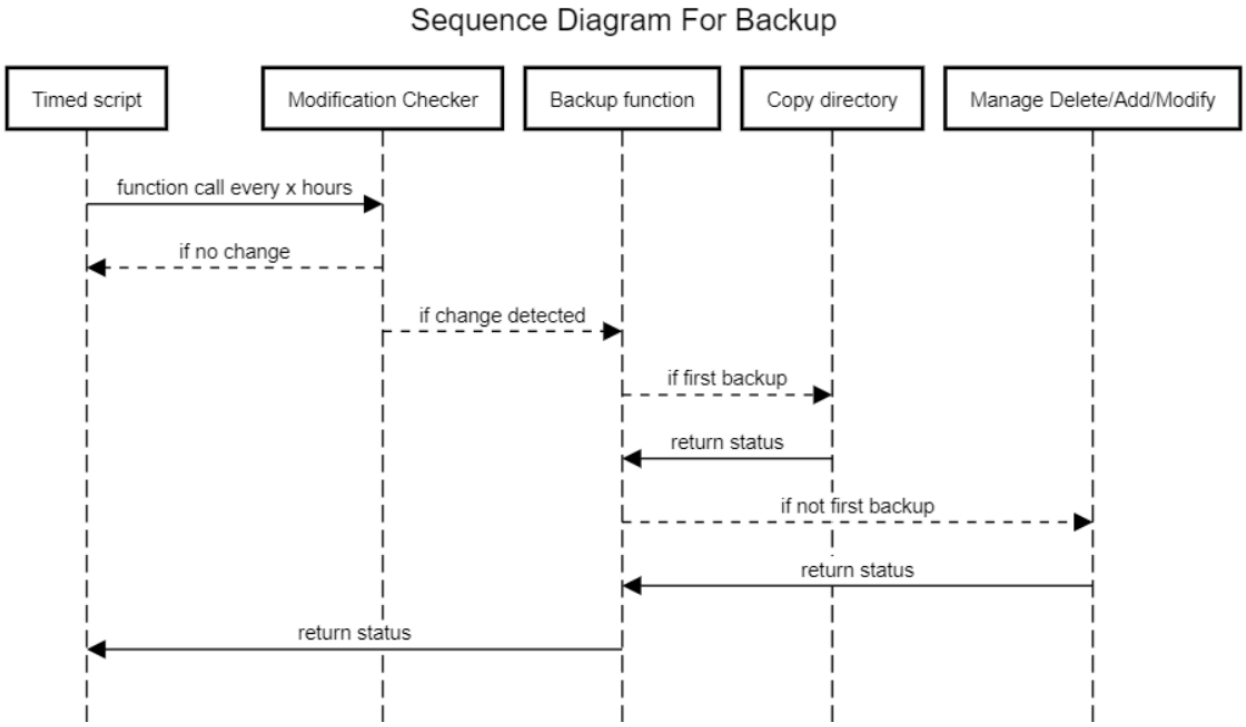
## Components

- A timed script running every x hours to check for changes in dataset
- A checker function for detecting changes in dataset
- A backup function
- A checker each for finding deleted files, added files, and modified files
- A recovery function

# Workflow

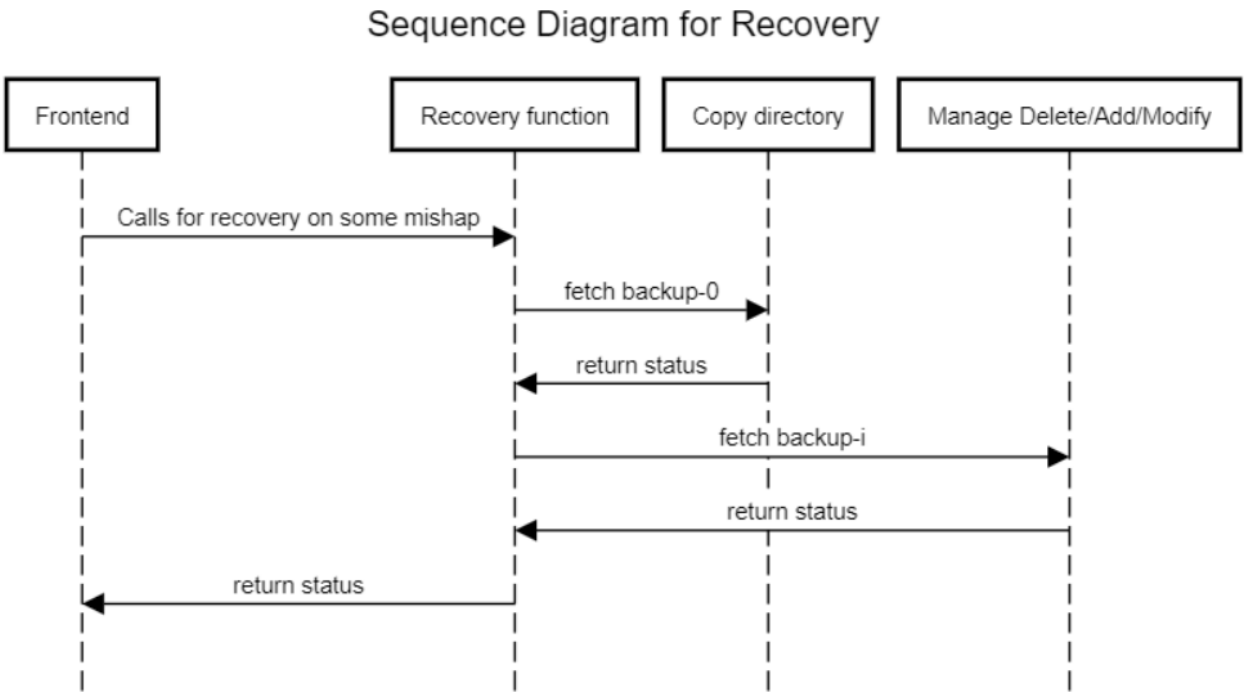
## Backup

- Timed script calls the checker function every x hours.
- If no change is found it does nothing and loops back.
- If there is a change found, it calls the backup function.
- The backup function checks if there have been any previous backups made.
- If not, it directly copies the dataset directory.
- If yes, then it creates a new empty folder named backup-i (i is the backup version index) then it calls checkers for deleted, added and modified files.
- The filenames are stored in respective txt files, and modified/added files are stored in respective folders, all inside the folder backup-i.




## Recovery

- When the recovery function is called (presumably by some frontend application), it starts from backup-0, copies it, the based on consequent backup versions, it replaces modified files, adds added files, and removes deleted files.
- This is iterated until the timestamp of the backup version i is older than the requested backup timestamp.




In the future, if space constraints are loosened, and faster recovery is needed, we can store full copies of the dataset stored at some of the intermediate backup versions (not just backup-0).

**System specification:** Any linux/windows based system would work as the backup server.

**High-level API calls** are “backup” and “recovery” functions.

The **development model** we follow would be the Iterative Model.

## Tools/Libraries

Python inbuilt libraries:

- os
- datetime
- time
- shutil
- time
- glob

Other libraries:

- ftplib (if file transfer across servers is required)
- pyzipper (if compression/encrypted-compression is required)